USB Device Notification

BACKGROUND OF THE INVENTION

Field of the Invention

The invention relates generally to communication in information processing systems and more specifically to notifying parts of systems that changes in other parts of the system have occurred.

Description of the Related Art

A recent advance in the design of computers involves hot-pluggability, the ability to connect a device to a computer bus while the computer is operating, and have that device communicate properly with the computer. In the past, computers could only be reliably connected to devices when power was not supplied. Now, busses have been designed which function properly when devices are connected or powered-up after the bus has power.

Unfortunately, solving the electrical problems associated with hot-pluggability does not guarantee that the device will communicate with the computer. The computer must be made aware that the device has been connected, and must be made aware of how that device communicates. This problem is typically solved either by having the computer poll the newly connected device for this information, or by having the newly connected device announce its presence.

Even this does not solve all problems involved with connecting a device to a computer after the computer starts operating. Clients, such as application programs and other routines may be established on the computer, as background processes, foreground processes, or as quiescent routines awaiting activation. If these clients have already determined which devices are connected before the newly connected device is connected to the bus, the clients may not be able to take advantage of the presence of the newly connected device. Therefore, what is needed is a method or apparatus for

notifying clients of the presence of the newly connected device. Furthermore, not all application programs or routines may want to know of specific devices being connected, so what is also needed is a method or apparatus for selectively notifying clients of the presence of newly connected devices.

SUMMARY OF THE INVENTION

The present invention, in one embodiment, is a method of notifying clients of a change in a system including a client requesting notification of a change in the system, detecting the change in the system, and notifying the client requesting notification that the change in the system occurred. The present invention may further include maintaining a list of requests for notification and removing requests for notification when a client terminates a request for notification.

The invention, in an alternate embodiment, is a subsystem for notifying clients of a change in a system including means for a client to request notification of the change in the system, means for detecting the change in the system, and means for notifying the client requesting notification that the change in the system occurred.

The present invention may be implemented in a system including a processor and memory. Likewise, the present invention may be embodied in instructions encoded in a computer readable medium. Furthermore, the present invention may be practiced in a system employing a USB.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the accompanying figures.

Figure 1a illustrates a system suitable for use with the present invention.

Figure 1b illustrates an alternate conception of the system of Figure 1a.

Figure 2 illustrates another system suitable for use with the present invention.

Figure 3 illustrates a process followed in the practice of one embodiment of the present invention.

Figure 4 illustrates one representation of communication that may occur in practicing the present invention.

Figure 5 illustrates an example of a machine-readable medium in accordance with the present invention.

DETAILED DESCRIPTION

A method and apparatus for USB Device Notification is described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the invention. It will be apparent, however, to one skilled in the art that the invention can be practiced without these specific details. In other instances, structures and devices are shown in block diagram form in order to avoid obscuring the invention.

Reference in the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the invention. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment.

Figure 1a illustrates a system suitable for use with the present invention. The system includes Host 100, HDD (Hard Disk Drive) 110, Keyboard 120, all connected to Bus 130. Additionally, Printer 140 may be connected to Bus 130, thereby allowing communication between Host 100 and Printer 140. Figure 1b illustrates an alternate conception of the system of Figure 1a. Here, the connections are formed with a star topology as typically used when making connections via the Universal Serial Bus (USB). There is a direct connection between Host 100 and Keyboard 120, between Host 100 and HDD 110, and between Host 100 and Hub 135. Printer 140 is optionally connected to Hub 135. In this example, the connections between the devices are the physical connections that substitute for the logical Bus 130. Further information on the USB may be obtained by consulting the USB Specification 1.0, January 15, 1996, which document is hereby incorporated by reference.

Host 100 may be a computer or other processing apparatus suitable for connection to peripheral devices. If a computer, Host 100 includes a microprocessor

and may also include some form of internal storage. However, Host 100 may be a controller which serves as a gateway for signals to and from the devices connected thereto.

Figure 2 illustrates another system suitable for use with the present invention. Microprocessor 210 is coupled to Bus 230, which is coupled to both Memory 220 and USB Control 240. USB Control 240 is coupled to HDD 260, Modem 270 and Printer 280 through USB 250. In a typical USB implementation, USB Control 240 is a control hub that is directly connected to each of HDD 260, Modem 270, and Printer 280. Since the USB allows for hot insertion and removal, Modem 270 may be connected to USB Control 240 after Microprocessor 210 is powered, or Modem 270 may be disconnected from USB Control 240 while Microprocessor 210 is still powered.

Microprocessor 210 communicates with each of the devices coupled to USB Control 240, so Microprocessor 210 must be kept apprised of what is connected to USB Control 240. Otherwise, Microprocessor 210 may attempt to send instructions to Modem 270 after Modem 270 has been disconnected from USB Control 240. Likewise, Microprocessor 210 may need to access HDD 260. If HDD 260 was not connected to USB Control 240 when Microprocessor 210 initially sought access to HDD 260, Microprocessor 210 may fail to complete its processing due to its inability to access HDD 260.

As mentioned above, Host 100 may be a computer or other processing apparatus suitable for connection to peripheral devices. In one embodiment, Host 100 includes Microprocessor 210, Memory 220, Bus 230 and USB Control 240. Microprocessor 210 may be a processor such as those manufactured by Motorola or Intel. Memory 220 may be dynamic random access memory (DRAM), and may also include static RAM or ROM of some form. Note that Memory 220, Microprocessor 210 and Bus 230 may be incorporated on a single device, or each may be composed of several individual devices. Typically, Host 100 also includes some form of input device,

such as a keyboard, pointing device, or scanner, and some form of output device, such as a display or speaker. Host 100 may also be connected to input devices such as cameras, storage media, or microphones, and may also be connected to output devices such as printers or storage media. Host 100 may further be connected to such input/output devices as modems or touch-sensitive screens, too. Storage media (machine-readable media) may include magnetic disks, carrier waves, optical disks, magnetic tape, memory as described with regard to Memory 220 above, and other forms of media suitable for storage of information.

It will be appreciated that Host 100 may take on other forms, such as a network computer or intelligent appliance. Typically, Host 100 will be controlled by an operating system, such as MacOS available from Apple Computer Corp. of Cupertino, California, the Linux operating system, or Windows '95 available from Microsoft Corporation of Redmond, Washington. Under control of such an operating system, Host 100 will execute programs such as application programs. Likewise, the operating system will include routines. These routines and programs will, at times, access peripheral devices coupled to Host 100. When these routines and programs access these peripheral devices, adequate information on which peripheral devices are currently coupled to Host 100 and what other changes have occurred in the system is essential.

These routines and programs may be thought of as clients for purposes of discussion. A method of informing these clients of what changes have occurred in the system will now be explained. Figure 3 illustrates a process followed in the practice of one embodiment of the present invention, whereby the clients are notified of changes in the system. First, a client requests notification of certain changes in Request Entry 310. As a result of that request, Parameter Storage 320 occurs, storing parameters of the request submitted by the client. Later, the client may no longer need information on changes, and therefore will terminate the previous request for notification at Request Termination 330. As a result of the termination, Parameter Removal 340 occurs, and

the parameters stored relating to the terminated request are removed. Requests may, in one embodiment, be submitted to a Device Notification routine, which will then either store or remove the parameters of the requests as appropriate.

Simultaneous with the operation on requests, the Device Notification routine will also determine whether changes are occurring in the system. When a change is detected as in Change Detection 360, the Device Notification routine acts upon that change. In Determination 370, the Device Notification routine will determine which requests have parameters that include the change detected in Change Detection 360. The Device Notification routine will then proceed to Notification 380, that is notifying each client that submitted a request with parameters that include the detected change. Both cycles Request Entry 310 through Parameter Removal 340 and Change Detection 360 through Notification 380 proceed independently of each other.

Figure 4 illustrates one representation of communication that may occur in practicing the present invention. Client 410 communicates with Device Notification routine 420, such as through requests for notification and termination of these requests. These requests may, in one embodiment, include an Installed Routine 430, which Device Notification 420 may activate in order to notify Client 410 of a detected change. Such an Installed Routine 430 may communicate with both Client 410 and Device Notification routine 420. Device Notification 420 communicates with Lower Level 440 to detect changes in the system, and then compares those changes with the requests for notification received from Client 410 to determine whether Client 410 needs notification of the detected changes. Lower Level 440 may be a part of an operating system, implemented in either hardware, software, or firmware. Alternatively, Lower Level 440 may be a physical device such as USB Control 240. Changes in the system may include connection or disconnection of a device such as a printer, errors in interfacing or communicating with a device, or a device being busy due to previous activity.

In one embodiment, utilized in implementing the present invention in conjunction with the MacOS operating system available from Apple Computer, Corp. of Cupertino, California, communication between a Client and a Device Notification routine occurs when the Client calls a `USBInstallDeviceNotification` function. Such a function call serves as a request for notification. A call to such a function looks like:

```
void USBInstallDeviceNotification(
                        USBDeviceNotificationParameterBlock *pb);
```

pb — A pointer to the `USBDeviceNotificationParameterblock`

The `USBDeviceNotificationParameterblock` includes information on what types of changes the Client wants notification of and what routine (Installed Routine) to use to communicate with the Client. The `USBDeviceNotificationParameterblock` is defined as:

```
/* Device Notification Parameter Block */
struct USBDeviceNotificationParameterBlock
(
UInt16                                  pbLength;
UInt16                                  pbVersion;
USBNotificationType                     usbDeviceNotification
UInt8                                   reserved1;
USBDeviceRef                            usbDeviceRef;
UInt16                                  usbClass;
UInt16                                  usbSubClass;
UInt16                                  usbProtocol;
UInt16                                  usbVendor
UInt16                                  usbProduct
OSStatus                                result;
UInt32                                  token;
USBDeviceNotificationCallbackProcPtr    callback;
UInt32                                  refcon;
};
```

Field descriptions

`<--> usbDeviceNotification`	The type of notification
	The following notifications are defined:
	`kNotifyAnyEvent`
	`kNotifyAddDevice`
	`kNotifyAddInterface`
	`kNotifyRemoveDevice`
	`kNotifyRemoveInterface`

<-- usbDeviceRef	The device reference for the target device
<--> usbClass	The class of the target device, use kUSBAnyClass for any class
<--> usbSubClass	The Subclass of the target device, use kUSBAnySubclass for any subclass
<--> usbProtocol	The protocol of the target device, use kUSBAnyProtocol for any protocol
<--> usbVendor	The Vendor ID of the target device, use kUSBAnyVendor for any vendor
<--> usbProduct	The product ID of the target device, use kUSBAnyProduct for any product
<-- result	The status of the call
<-- token	The identifier for this notification request
--> callback	A pointer to the callback routine to be called when the notification criteria is satisfied

Note that an arrow '-->' indicates a parameter sent from the Client to the Device Notification routine, and an arrow '<--' indicates a parameter sent from the Device Notification routine to the Client. A double-ended arrow '<-->' indicates a parameter communicated in both directions. Also, note that the parameters sent from the Client to the Device Notification routine specify what type of changes the Client seeks notification of. A first request from a first Client might seek notification of the connection of any printer. A second request, from either a first Client or a second Client, may seek notification of connection of any printer manufactured by a particular vendor, such as Hewlett-Packard. The first request would simply specify a printer as the class of device in the usbClass field. The second request would specify a printer in the usbClass field, and also specify the vendor Hewlett-Packard in the usbVendor field.

As will be appreciated, requests for notification may be specific or general, and multiple requests may be made by a single Client, such as a Client seeking access to

both a modem and a printer. In one embodiment, the Device Notification routine provides notification by calling a callback routine such as the one supplied in the `callback` field. Such a routine would be declared as:

```
typedef void (USBDeviceNotificationCallbackProc)
                        (USBDeviceNotificationParameterBlockPtr pb);

typedef USBDeviceNotificaitonCallbackProc
                        *USBDeviceNotificationCallbackProcPtr;
```

This implementation provides flexibility to the Client, allowing the Client to receive notification of changes in a manner useful to the Client. In particular, the Client may supply a routine as a callback routine which alerts it to the presence of a newly connected device, or it may supply a routine that adds a newly connected device to a list of devices maintained by the Client. Likewise, notification of removal or disconnection of a device may result in calling a different routine supplied in a different request.

Since the Device Notification routine receives multiple requests, it must maintain a list of each request and its associated parameters. Such a list would include the parameters passed to the Device Notification routine when a client calls `USBInstallDeviceNotification` and a token identifying each request uniquely. However, if a Client such as an application program is terminating, the Client should terminate the request of notification, and it can do this with a call to a `USBRemoveDeviceNotification` function. Such a function is declared as:

```
OSStatus USBRemoveDeviceNotification (UInt32 token);
```

`token`	Notification identifier from the previously installed device notification routine.

Upon termination of the request, the information relating to that request that is maintained by the Device Notification routine is discarded, and the Device Notification routine no longer seeks to notify the Client in accordance with that request.

Turning to Figure 5, an example of a machine-readable medium in accordance with the present invention is illustrated. In one embodiment, Machine-Readable Medium 500 contains Operating System routines 520 embodying Device Notification 420 of Figure 4, Client 510, the routines that embody Client 410 of Figure 4, List of Requested Notices 530, and Callback routine 540, embodying Installed Routine 430 of Figure 4. Client 510 requests notification of changes in the status of the system. Operating System routines 520, such as those in the MacOS operating system available from Apple Computer, Inc. of Cupertino California, receive the requests for notification from Client 510, manage the List of Requested Notices 530 which stores the information on each request, detect changes in the status of the system, and call the Callback routine 540 when a status change corresponding to a request is detected or otherwise notify a requestor of the status change. List of Requested Notices 530 includes information on each request, such as that described in the above description of the invention as implemented in the MacOS operating system, and may include a pointer to Callback routine 540 or a similar callback routine for each individual request. Callback routine 540 calls Client 510 or otherwise notifies Client 510 of the change in status, and may be a portion of Client 510.

It will be appreciated that each of the above routines or portions of information may be stored in machine-readable media (or a single medium) in distributed or whole form. In either case, the information will typically be stored in a form suitable for execution (such as executable instructions for example) by a processor such as a microprocessor, or for use during execution by a processor. Additionally, the information, such as the List of Requested Notices 530, may be changed during execution, including creation or deletion of entries or creation or deletion of the entire List 530. In Figure 5, the appearance is created that everything is stored in integrated or whole form in a single machine-readable medium, but it will be appreciated that

storage in distributed form in one medium or over multiple media does not depart from the spirit of the invention.

In the foregoing detailed description, the method and apparatus of the present invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the present invention. The present specification and figures are accordingly to be regarded as illustrative rather than restrictive.